IN THE DRAWINGS:

Please approve the accompanying Request for Approval of Drawing Changes.

REMARKS:

After this response, claims 1 to 8 and 24 to 35 are pending. Claims 1 to 3 and 6 to 8 have been amended, claims 9 to 23 have been cancelled, and claims 24 to 35 have been added. Claims 1 and 26 are the independent claims. Reconsideration and further examination are respectfully requested.

Drawings

Applicants have requested approval of drawing changes and have amended the specification so as to address the drawing objections raised in the Office Action. Accordingly, withdrawal is respectfully requested of the drawing objections.

Abstract

The abstract has been amended to conform to the requirements set forth in the Office Action. Approval of the amended abstract is respectfully requested.

Incorporation by Reference

The Office Action required addition of the serial numbers to the descriptions on pages 8 and 9 of the documents incorporated by reference into the application. Applicants have so amended these descriptions. Accordingly, withdrawal is respectfully requested of the objection to the text at pages 8 and 9.

Claim Objection

The Office Action objected to claims 6, 13 and 21 for improper multiple dependencies. Applicants have corrected the multiple dependency in claim 6. Accordingly, withdrawal is respectfully requested of the objection to claim 6. Claims 13 and 21 have been cancelled, rendering the objection to those claims moot.

Claim Rejections

Claims 1 to 23 were rejected under 35 U.S.C. § 102(b) as allegedly anticipated by U.S. Patent No. 5,819,292 (Hitz). Applicants have cancelled claims 9 to 23, rendering the rejection of those claims moot. Claim 1 has been amended.

Claim 1 recites a method for capturing the contents of the files and directories in a file system. The file system includes a set of storage blocks in a mass storage system. The method includes the step of recording an active map in the file system of the storage blocks used by the active file system. The method also includes the step of recording a consistency point in the file system including a consistent version of the file system at a previous time. The

consistency point includes a copy of the active map at the previous time. The method also includes the step of refraining from writing data to storage blocks in response to the active map and at least one copy of the active map at the previous time.

The applied Hitz reference is not seen to disclose or to suggest the foregoing features of claim 1, at least with respect to refraining from writing data to storage blocks in response to an active map and at least one copy of the active map at a previous time.

In this regard, Hitz uses a block map ("blkmap") to determine which blocks are unallocated and therefore free for writing new data. This block map is discussed in Hitz at col. 9, lines 50 to 65. According to Applicants' understanding, the block map does not include and is not responsive to a copy of an active map at a previous time. Rather, in Hitz, the block map by itself indicates whether or not blocks are available for writing data.

Accordingly, Hitz is not seen to disclose or even to suggest claim 1's feature of refraining from writing data to storage blocks in response to an active map and at least one copy of the active map at a previous time. Withdrawal is therefore respectfully requested of the § 102 rejection of claim 1 and the claims that depend therefrom.

<u>New Claims</u>

New claims 24 and 25 concern use of a summary map that is generated responsive to at least one copy of an active map at a previous time. For example, as described at page 20, lines 1 to 8, of the application, the summary map can be generated by performing an inclusive

OR operation on plural such previous active maps. This feature is believed to even further distinguish the claimed invention from Hitz.

Claims 26 to 35 recite file systems that substantially implement the methods of claims 1 to 8, 24 and 25. These claims also are believed to be allowable over Hitz for at least the reasons set forth above.

Closing

In view of the foregoing amendments and remarks, the entire application is believed to be in condition for allowance, and such action is respectfully requested at the Examiner's earliest convenience.

Applicants undersigned attorney can be reached at (614) 486-3585. All correspondence should continue to be directed to the address indicated below.

Respectfully submitted,

Dated: February 4, 2003

Dane C. Butzer
Reg. No. 43,521

The Swernofsky Law Group
P.O. Box 390013
Mountain View, CA 94039-0013
(650) 947-0700

Changes to Specification

Pursuant to 37 C.F.R. § 1.121(b)(iii), changes to the specification effected by the accompanying paper are indicated below.

The paragraph at page 5, lines 1 to 12, has been amended as follows:

In a third aspect of the invention, the state of the active file system is described by a set of metafiles; in particular, a bitmap (henceforth the "active map") describes which blocks are free and which are in use by the active file system. The inode file describes which blocks are used by each file, including the metafiles. The inode file itself is described by a special root inode, also known as the "fsinfo block." This copy of the root inode becomes the root of the snapshot. The root inode captures all required states for creating the snapshot such as the location of all files and directories in the file system [, it]. During subsequent updates of the active file system, the system consults the bitmap included in the snapshot (the "snapmap") to determine whether a block is free for reuse or belongs to a snapshot. This mechanism allows the active file system to keep track of which blocks each snapshot uses without recording any additional bookkeeping information in the file system.

The paragraphs at page 8, line 16, to page 9, line 17, have been amended as follows:

- U.S. Patent Application Serial No. [__________] 09/642,063, Express Mail Mailing No. EL 524781089US, filed August 18, 2000, in the name of Blake LEWIS, attorney docket number 103.1033.01, titled "Reserving File System Blocks"

- U.S. Patent Application Serial No. [__________] 09/642,062, Express Mail Mailing No. EL524780242US, filed August 18, 2000, in the name of Rajesh SUNDARAM, attorney docket number 103.1034.01, titled "Dynamic Data Storage"

- U.S. Patent Application Serial No. [__________] 09/642,066, Express Mail Mailing No. EL524780256US, filed August 18, 2000, in the name of Ray CHEN, attorney docket number 103.1047.01, titled "manipulation of Zombie Files and Evil-Twin Files"

- U.S. Patent Application Serial [Number __________] No. 09/642,064, in the names of Scott SCHOENTHAL, Express Mailing Number EL524781075US, titled "Persistent and Reliable Delivery of Event Messages", assigned to the same assignee, attorney docket number 103.1048.01, and all pending cases claiming the priority thereof.

and

- U.S. Patent Application Serial [Number __________] 09/642,065, in the names of Douglas P. DOUCETTE, Express Mailing Number EL524781092US, titled "Improved Space Allocation in a Write Anywhere File System", assigned to the same assignee, attorney docket number 103.1045.01, and all pending cases claiming the priority thereof.

The paragraph at page 15, line 17, to page 16, line 6, has been amended as follows:

The active map 115 of the active file system 110 is a bitmap associated with the vacancy of blocks for the active file system 110. The respective snapmaps 140, 145, 150 and 155 are active maps that can be associated with particular snapshots 120, 125, 130 and 135. [and an inclusive OR] A summary map 160 is an inclusive OR of the snapmaps 140, 145, 150 and 155. Also shown are other blocks 115 including double indirect blocks 130 and 132, indirect blocks 165, 166 and 167 and data blocks 170, 171, 172 and 173. Finally, Figure 1 shows the spacemap 180 including a collection of spacemap blocks of numbers [181,] 182, [183,] 184, 186, 188 and 190.

The paragraph at page 17, line 17, to page 19, line 2, has been amended as follows:

Another inode block in the inode file 105 is inode block N 195 [212]. This block includes a set of pointers to a collection of snapshots 120, 125, 130 and 135 of the volume. Each snapshot includes all the information of a root block and is equivalent to an older root block from a previous active file system. The snapshot 120 may be created at any past CP. Regardless when the snapshot is created, the snapshot is an exact copy of the active file system at that time.

¶

The newest snapshot 120 includes a collection of pointers that are aimed directly or indirectly to the same inode file 105 as the root block 100 of the active file system 110. As the active file system 110 changes (generally from writing files, deleting files, changing attributes of files, renaming file, modifying their contents and related activities), the active file system and snapshot will diverge over time. Given the slow rate of divergence of an active file system from a snapshot, any two snapshots will share many of the same blocks.

¶

The newest snapshot 120 is associated with snapmap 140. Snapmap 140 is a bit map that is initially identical to the active map 115. The older snapshots 125, 130 and 135 [194] have a corresponding collection of snapmaps 145, 150 and 155. Like the active map 115, these snapmaps 145, 150 and 155 include a set of blocks including bitmaps that correspond to allocated and free blocks for the particular CP when the particular snapmaps 145, 150 and 155 were created.

¶

Any active file system may have a structure that includes pointers to one or more snapshots. Snapshots are identical to the active file system when they are created. It follows that snapshots contain pointers to older snapshots. There can be a large number of previous snapshots in any active file system or snapshot. In the event that there are no snapshot, there will be no pointers in the active file system.

The paragraph at page 20, lines 1 to 8, has been amended as follows:

A summary map 160 is created by using an IOR (inclusive OR) operation 139 on the snapmaps 140, 145, 150 and 155. Like the active map 115 and the snapmaps 140, 145, 150 and 155, the summary map 160 is a file whose data blocks (1, 2, 3, ...Q) contained a bit map. Each bit in each block of the summary map [.] describes the allocation status of one block in the system with "1" being allocated and "0" being free. The summary map 160 describes the allocated and free blocks of the entire volume from all the snapshots 120, 125, 130 and 135 combined. The use of the summary file 160 is to avoid overwriting blocks in use by snapshots.

The paragraphs at page 20, line 10, to page 21, line 15, have been amended as follows:

An IOR operation on sets of blocks (such as 1,024 blocks) of the active map 115 and the summary map 160 produces a spacemap 180. Unlike the active map 115 and the summary map 160, which are a set of blocks containing bitmaps, the spacemap 180 is a set of blocks including [181,] 182, [183,] 184, <u>186, 188</u> and 190 containing arrays of binary numbers. The binary numbers in the array represent the addition of all the vacant blocks in a region containing a fixed number of blocks, such as 1,024 blocks. The array of binary numbers in the single spacemap block 181 represents the allocation of all blocks for all snapshots and the active file system in one range of 1,024 blocks. Each of the binary numbers [181,] 182, [183,] 184, <u>186, 188</u> and 190 in the array are a fixed length. In a preferred embodiment, the binary numbers are 16 bit numbers, although only 10 bits are used.

In a preferred embodiment, the large spacemap array binary number 182 (0000001111111110=1,021 in decimal units) tells the file system that the corresponding range is relatively full. In such embodiments, the largest binary number 00001111111111 (1,023 in decimal) represents a range containing at most one empty [.] block. The small binary number 184 (0000000000001110=13 in decimal units) instructs the file system that the related range is relatively empty. The spacemap 180 is thus a representation in a very compact form of the allocation of all the blocks in the volume broken into 1,024 block sections. Each 16 bit number in the array of the spacemap 180 corresponds to the allocations of blocks in the range containing 1,024 blocks or about 4 MB. Each spacemap block 180 has about 2,000 binary numbers in the array and they describe the allocation status for 8 GB. Unlike the summary map 120, the spacemap block 180 needs to be determined whenever a file needs to be written.

The paragraphs at page 22, line 6, to page 23, line 14, have been amended as follows:

The snapmap 205 of the previous active file system, snapshot #1 201, is a bitmap associated with the vacancy of blocks for snapshot #1 201. The respective snapmaps 225, 230 and 235 are earlier active maps that can be associated with particular snapshots 210, 215 and 220. [and an inclusive OR] A summary map 245 is an inclusive OR of the snapmaps 225, 230 and 235. Also shown are other blocks 211 including double indirect blocks 240 and 241 [332], indirect blocks 250, 251 and 252, and data blocks 260, 261, 262, and 263 [and 264]. Finally,

Figure 2 shows the spacemap 270 of snapshot #1 201 including a collection of spacemap blocks of binary numbers [272, 273, 274, 275 and 276].

The old root block 200 includes a collection of pointers that were [are] written to the previous active file system when the system had reached the previous CP. The pointers are aimed at a set of indirect (or triple indirect, or double indirect) inode blocks (not shown) or directly to the inode file 202 consisting of a set of blocks known as inode blocks 281, 282, 283, 284 and 285.

An inode block 281 in the inode file 202 points to other blocks 328 in the old root block 200 [201] starting with double indirect blocks 240 and 241 [332] (there could also be triple indirect blocks). The double indirect blocks 240 and 241 [332] include pointers to indirect blocks 250, 251 and 252. The indirect blocks 250, 251 and 252 include pointers that are directed to data leaf blocks 260, 261, 262, and 263 [and 264] of the snapshot #1 [active file system] 201.

Inode block 283 in the inode file 202 points to a set of blocks (1, 2, 3, ..., P) called the snap map 205. Each block in the snap map 205 is a bitmap where each bit corresponds to a block in the entire volume. A "1" in a particular position in the bitmap correlates with a particular allocated block in the snapshot #1 [active file system] 201. Conversely, a "0" correlates to the particular block being free for allocation in the old root block 200 [201]. Each block in the snap map 205 can describe up to 32K blocks or 128 MB.

The paragraph at page 24, lines 3 to 6, has been amended as follows:

Snapshot #1 201 also includes an old summary map 245 and old spacemap blocks 270. Although these blocks of data are included in snapshot #1 201 and previous snapshots, in a preferred embodiment, this data is not used by the active file system [of figure 2].

The paragraphs at page 24, line 13, to page 25, line 10, have been amended as follows:

A method 300 is performed by the file system 110 [100]. Although the method 300 [400] is described serially, the steps of the method 300 can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method 300 be performed in the same order in which this description lists the steps, except where so indicated.

At a flow point 305, the file system 110 [100] is ready to perform a method 300.

At a step 310, a user will request a snapshot of the file system 110 [100].

At a step 315, a timer associated with the file system 110 [100] initiates the creation of a new snapshot.

At a step 320, the file system 110 [100] receives a request to make a snapshot.

At a step 325, the file system 110 [100] creates a new file.

The paragraphs at page 25, line 15, to page 26, line 3, have been amended as follows:

At a step 335, the file system 110 [100] makes the file read only.

At a step 340, the file system 110 [100] updates the new summary map by using an inclusive OR of the most recent snapmap and the existing summary file. This step must be done before any blocks are freed in the corresponding active map block. If multiple snapshots are created such that the processing overlaps in time, the update in step 340 need only be done for the most recently created snapshot.

The abstract has been amended as follows:

[The invention provides an] An improved method and apparatus for creating a snapshot of a file system. [In a first aspect of the invention, a "copy-on-write" mechanism is used. An effective snapshot mechanism must be efficient both in its use of storage space and in the time needed to create it because file systems are often large. The snapshot uses the same

blocks as the active file system until the active file system is modified. Whenever a modification occurs, the modified data is copied to a new block and the old data is saved (henceforth called "copy-on-write"). In this way, the snapshot only uses space where it differs from the active file system, and the amount of work required to create the snapshot is small. In a second aspect of the invention, a] A record of which blocks are being used by a [the] snapshot is included in the snapshot itself, allowing effectively instantaneous snapshot creation and deletion. [In a third aspect of the invention, the] The state of the active file system is described by a set of metafiles; in particular, a bitmap (henceforth the "active map") describes which blocks are free and which are in use. The inode file describes which blocks are used by each file, including the metafiles. The inode file itself is described by a special root inode, also known as the "fsinfo block". The system begins creating a new snapshot by making a copy of the root inode. This copy of the root inode becomes the root of the snapshot. [The root inode captures all required states for creating the snapshot such as the location of all files and directories in the file system, it. During subsequent updates of the active file system, the system consults the bitmap included in the snapshot (the "snapmap") to determine whether a block is free for reuse or belongs to the snapshot. This mechanism allows the active file system to keep track of which blocks each snapshot uses without recording any additional bookkeeping information in the file system. In a fourth aspect of the invention, a snapshot can also be deleted instantaneously simply by discarding its root inode. Further bookkeeping is not required, because the snapshot includes it's own description. In a fifth aspect of the invention, the performance overhead associated with the search for free blocks is reduced by the inclusion of a summary file. The summary file identifies

blocks that are used by at least one snapshot; it is the logical OR of all the snapmap files. The write allocation code decides whether a block is free by examining the active map and the summary file. The active map indicates whether the block is currently in use in the active file system. The summary file indicates whether the block is used by any snapshot. In a sixth aspect of the invention, the summary file is updated in the background after the creation or deletion of a snapshot. This occurs concurrently with other file system operations. Two bits are stored in the file system "fsinfo block" for each snapshot. These two bits indicate whether the summary file needs to be updated using the snapshot's snapmap information as a consequence of its creation or deletion. When a block is freed in the active file system, the corresponding block of the summary file is updated with the snapmap from the most recently created snapshot, if this has not already been done. An in-core bit map records the completed updates to avoid repeating them unnecessarily. This ensures that the combination of the active bitmap and the summary file will consistently identify all blocks that are currently in use. Additionally, the summary file is updated to reflect the effect of any recent snapshot deletions when freeing a block in the active file system. This allows reuse of blocks that are now entirely free. After updating the summary file following a snapshot creation or deletion, the corresponding bit in the fsinfo block is adjusted. In a seventh aspect of the invention, the algorithm for deleting a snapshot involves examining the snapmaps of the deleted snapshot and the snapmaps of the next oldest and next youngest snapshot. A block that was used by the deleted snapshot but is not used by its neighbors can be marked free in the summary file, as no remaining snapshot is using it. However, these freed blocks cannot be reused immediately, as the snapmap of the deleted

snapshot must be preserved until summary updating is complete. During a snapdelete, free blocks are found by using the logical OR of the active bitmap, the summary file, and the snapmaps of all snapshots for which post-deletion updating is in progress. In other words, the snapmap of the deleted snapshot protects the snapshot from reuse until it is no longer needed for updating. In the preferred embodiment, the invention is operative on WAFL file system. However, it is still possible for the invention to be applied to any computer data storage system such as a database system or a store and forward system such as cache or RAM if the data is kept for a limited period of time.]

Changes to Claims

Pursuant to 37 C.F.R. § 1.121(c)(ii), changes to any claims effected by the accompanying paper are indicated below.

Claims 1 to 3 and 6 to 8 have been amended as follows:

1. (Amended) A method for capturing the contents of the files and directories in a file system, said file system comprising a set of storage blocks in a mass storage system including steps of [for]

recording an active map in said file system of said storage blocks used by said active file system [not available for writing data];

recording a consistency point in said file system including a consistent version of said file system at a previous time, said consistency point including a copy of said active map at said previous time; and

refraining from writing data to storage blocks in response to said active map [;] and at least one [of] said copy of said active map at said previous time.

2. (Amended) A method as in claim 1, wherein said step of [for] refraining includes determining a logical union of said storage blocks used by one or more of said copies of said active map at said previous time.

3. (Amended) A method as in claim 1, wherein said step <u>of</u> [for] refraining includes determining a subset of said storage blocks used by one or more of said copies of said active map at said previous time.

6. (Amended) A method as in claim 1 [and] <u>or claim</u> 5, including <u>the step of</u> removing a root inode of said snapmap using a snap delete.

7. (Amended) A method as in claim 6, <u>further</u> including steps <u>of</u> [for] determining not to write to a block after said step <u>of removing</u>, provided <u>a</u> [the] previous or next snapmap uses said block.

8. (Amended) A method as in claim 1, <u>further</u> including <u>steps of</u> [a copy-on-write mechanism for] copying modified data to a new block and saving old data in a current data block <u>so as to implement a copy-on-write mechanism</u>.

Claims 9 to 23 have been deleted, and claims 24 to 35 have been added.